

<Draw It or Lose It>

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <May/20/2025> | <Angel Esparza> | <Created Executive Summary, Design Constraints, and created Domain Model.> |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

<The client is looking to make their app, “Draw It or Lose It” (based on the 1980’s televisions game) into a web-based game versus mobile app. The game consists of teams competing to guess what is being drawn. The client is not able to set up the environment of the game onto a web-based platform, which is where we come in. We’ll be focused on developing the android-app based game onto the web. Which will lead to higher accessibility amongst a global population of users.>

## Requirements

*<*The requirements of the game include: multiple team involvement in within the game, with multiple players assigned per team, the necessity of unique names for game and team names (to make sure no repetitious names are used), also only “one instance of the game” can be allowed at a time (unique attributes for each instance of game/team/player will be necessary.) >

## [Design Constraints](#_2et92p0)

<One of the biggest design constraints that comes to mind with this project is multiplatform development, especially in regard to web-based development. Being able to transition the android app to the webpage can cause many technical issues in adapting to the web software and interface. Another design constraint/worry is of sticking to the limitation of one instance of the game at a time.>

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

<As we can see Entity Class is the parent class and is the class of Game, Team, and Player (as its connected to the three of them). The three classes Game, Team, and Player are child classes of Entity Class thus maintaining all its attributes. The GameService class has its constructors set to private (in order to ensure no creation of instances following clients request). Game Service class also has the “addGame” method for unique game names and Game class has “add team” method for team names; to follow the clients request of the necessity of unique names for game and team names. The ProgramDriver class is connected to Singelton Tester class and has the “+main()” method, meaning this is where the main function is. The Program Driver class is dependent on the SingletonTester class. Which contains the singleton instance creation, to ensure only one instance (for GameService class) is used. Game and Team class also both have lists, with Game having Team list and Team Player list. Player Class sets for ensuring players maintain a unique id (has no list).

Now for how the object-oriented principles are shown. As for inheritance, it is the technique of re-using and extension of code and inheriting other code (thus the name). We see this is shown as Game, Team, and Player class extend to Entity class. As for encapsulation, it’s used as getter and setter methods are used (“-” and “+”) restricting certain access to some object components. As for abstraction, it was shown in this diagram as accessing the constructor directly isn't possible while still being able to create Team object (for data privacy) through use of list of existing games. Lastly, polymorphism is shown as “getGame” operated on multiple types of objects.>

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macs accessibility on the server end is much more accessible. With a very effective ecosystem overall, that is also flexible. The weakness on the server side of Mac is the limited options for overall server options. | Some pros are that the cost is better than its competitors, and it has high scalability and flexibility with software/tools. A weakness is the harder navigation on the server side. | As for advantages on Windows, it's very broad potential for server ecosystems, while also being very user/beginner friendly. As for weaknesses for Windows is that it’s a lot more expensive and has higher chances of security vulnerabilities. | As for an advantage; it’s more accessible for the average person due to its remote development capabilities (from size convenience). Disadvantages for Mobile Devices include inefficiency for multi-server usage and overall hardware would be much more limited compared to its competitors. It should also be noted; it varies most from device to device. |
| **Client Side** | It is a very easily manageable interface and aesthetically pleasing. You will also have significantly better time as well if you have other apple-based products, as its seamless connection to the Mac (with other apple devices) is very intuitive. Higher end user cost. Users will be using MacOS with any Mac computer. | Something to be noted on the client side for Linux is its much harder learning curve in comparison to its peers. But as for a pro, it should be noted; it is better on the cost side, compared to other options. | Easier learning curve and setup compared to its peers. Also is one of, if not the, most popular operating systems which can account for better market opportunities. | Lots of flexibility and important considerations overall. The most versatile and flexibility. Costs can lead to being more expensive compared to their peers, due to the end of development on different mobile platforms. |
| **Development Tools** | Relevant languages include CSS, HTML, and JavaScript. PyCharm. Also, Visual Studios work very well as a tool. And XCode will work as an efficient IDE for Mac. | Languages that work well are Python, C++, and Java Script. PyCharm works well too for a Linux IDE. | Languages that work well for Windows are HTML and C++ as well as Java Script. Lastly, Visual Studio will be a good IDE for Windows. | Java Script is an efficient language for mobile development. Xcode is a viable IDE for mobile development. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: One of the best operating platforms for The Gaming Room is the Windows operating system. Some of its positives include its larger userbase compared to other operating systems. Also, the varying selection of tools Microsoft/Windows maintains for overall development such as varying IDE’s and Code Editors made it an easy choice as an Operating Platform (Adekotujo, 2020).
2. **Operating Systems Architectures**: After researching Serverless architecture I have found it quite interesting and I’m looking to select it as the architecture for The Gaming Room project. Serverless Architecture is a software architecture concept where we can build projects and applications with no fundamental type of infrastructure (Bashir, 2019). As Serverless Architecture approach is Serverless, it allows for seamless and indefinite scaling, easier setup of environments, and reduced costs (Bashir, 2019). Windows contains varies Serverless Architecture options such as “Azure Functions and AWS Lambda” that I can use for “The Gaming Room”.
3. **Storage Management**: Since we’ll be operating on Windows, focusing on services such as Microsoft Azure can be beneficial. It’s an affordable cost of the service while also allowing for efficient scalability and durability. Azure contains multiple services that accommodate storage management (Collier, 2015). There are also other features within Windows such as “Storage Sense” that work is efficient in storage management. Allowing the user to manage their storage and files at their own discretion.
4. **Memory Management**: Windows 11 allows for management of memory through its accessible storage. The varies images for this game will have to be allocated to the proper locations that we are aware of. It helps to be aware of where certain images/files are being stored for no confusion.
5. **Distributed Systems and Networks**: Focusing on Cross-Platform development and network connectivity. This will allow for an easier transitional process to communicate between various platforms. For if the games focus on uses of network connectivity it will allow players across the world to communicate and play amongst each other. This will require being aware of possible network issues and errors that can arise.
6. **Security**: Security protocols will need to be established early. Various antivirus services will come at a cost but one that will be worth it when leading to protecting our information amongst various platforms. It is worth noting though that we will be focusing on using the Windows Operating system, which already comes with built-in anti-virus software. Such as “Windows Defender”.

References

Adekotujo, A., Odumabo, A., Adedokun, A., & Aiyeniko, O. (2020). A comparative study of operating systems: Case of windows, unix, linux, mac, android and ios. *International Journal of Computer Applications*, *176*(39), 16-23.

Bashir, F. (2019, July 5). *What is Serverless Architecture? What are its Pros and Cons?* FreeCodeCamp.org. <https://www.freecodecamp.org/news/what-is-serverless-architecture-what-are-its-pros-and-cons/>

Collier, M., & Shahan, R. (2015). *Microsoft azure essentials-fundamentals of azure*. Microsoft Press.